Characterizing the Impacts of Semi-supervised Learning for Weak Supervision

Jeffrey Li [1] **Jieyu Zhang** [1] **Ludwig Schmidt** [1] **Alexander Ratner** [1]

Abstract

Labeling training data is a critical and expensive step in producing high accuracy ML models, whether training from scratch or fine-tuning. To make labeling more efficient, two major approaches are programmatic weak supervision (WS) and semi-supervised learning (SSL). More recent works have either explicitly or implicitly used techniques at their intersection, but in various complex and ad hoc ways. In this work, we define a simple, modular design space to study the use of SSL techniques for WS more systematically. Surprisingly, we find that fairly simple methods from our design space match the performance of more complex state-of-the-art methods, averaging a 3 p.p. increase in accuracy/F1-score across 8 standard WS benchmarks. Further, we provide practical guidance on when different components are worth their added complexity and training costs. Contrary to current understanding, we find using SSL is *not* necessary to obtain the best performance on most WS benchmarks but is more effective when: (1) end models are smaller, and (2) WS provides labels for only a small portion of training examples.

1. Introduction

Learning with limited labels is a fundamental challenge in machine learning (ML) applications (Kocoń et al., 2023; Zhu et al., 2023). To address the significant costs of hand-labeling training sets, *programmatic weak supervision (WS)* has emerged as a high impact research area (Ratner et al., 2016; Zhang et al., 2022a), where the aim is to learn from multiple cheaper sources of *noisy labels*. Meanwhile, *semi-supervised learning (SSL)* is a more classical direction with similar high-level motivations. Instead of generating larger quantities of noisy labels, SSL aims to directly leverage additional *unlabeled data*. It seems natural that these two fields could be applied productively with one another, yet their intersection has not been systematically studied.

In this work, we anchor on WS approaches and study whether they can be enhanced using techniques from SSL. At a high-level, most WS methods consist of two steps. First, a *label model* aggregates a set of weak label sources to noisily label a training set. Commonly, these sources take the form of user-written heuristics (e.g., for sentiment analysis, a user may check for the keyword "great" to provide the label "positive"). Second, an *end model* is learned on this training set. Crucially, weak sources can often abstain (e.g., the absence of "great" may not imply "negative"), leaving certain examples to remain unlabeled and thereby unused. This presents a natural opportunity to use SSL.

Indeed, this approach of using SSL in WS settings has motivated several recent methods (Yu et al., 2021; Ren et al., 2020; Karamanolakis et al., 2021; Gao et al., 2022). Though these works often attribute their observed improvements to their usage of unlabeled data (Zhang et al., 2021; Yu et al., 2021; Ren et al., 2020), they also incorporate various algorithmic components in addition to SSL. Thus, we lack clarity about the precise contributions of SSL and whether simpler methods might also suffice. Also, while varying the amount of unlabeled data is crucial when evaluating SSL methods (Oliver et al., 2018), previous WS benchmarks contain little diversity along this key dimension: most leave only a small minority of examples as unlabeled. Here, we conduct a more systematic study of how useful SSL is in a variety of WS settings, as well as how and when to best employ it.

Specifically, we first organize the intersection between SSL and WS by proposing an explicit design space, centered around disentangling the following key methodological considerations:

(1) *Thresholding: What to treat as (un)labeled?* Because WS uses heuristics for labeling, it often can only provide labels for a subset of examples, leaving the rest as unlabeled. Further, it can be beneficial to *additionally* remove some (likely) incorrect labels provided by WS, as shown by Lang et al. (2022). Since unlabeled data can result in multiple ways when using WS, we view "what to treat as unlabeled" as a non-trivial and first-class axis in our design space.

[1]University of Washington. Correspondence to: Jeffrey Li <jwl2162@cs.washington.edu>.

Workshop on Data-centric Machine Learning (DMLR) at the 40^{th} International Conference on Machine Learning, 2023.

(2) *SSL Technique: How to use unlabeled examples*? After deciding what data should be treated as unlabeled, one can leverage these examples by simply using any existing SSL technique. Most recent proposals do so via self-training, which uses the end model to periodically provide labels for unlabeled examples; in addition, we also try other out-of-the-box SSL methods.

(3) *Re-labeling: Whether to update weak labels during end model training?* Because some labels from WS are incorrect, it can be helpful to *re-label* a training set with the end model during training. This approach, increasingly used by WS methods (Yu et al., 2021; Karamanolakis et al., 2021; Cachay et al., 2021), is often packaged as part of self-training-based SSL. Here, we identify that re-labeling and SSL can be *independently* employed, and aim to disentangle their impacts.

With our design space, we can organize previous works and modularly generate a variety of methods. We test these methods on several standard WS benchmarks, finding that our design space is sufficient for matching the performance of more complex state-of-the-art methods. We then compare methods within our design space to ablate the importance of each axis and so provide guidance on when each is worth using. We summarize our key findings as follows:

- By searching over our design space, we identify two methods that at least match all previous baselines across 6 of 8 WS benchmarks, averaging a 3 p.p. increase in accuracy/F1-score.
- While previous works emphasize utilizing data left unlabeled by WS sources, we find that on 6 of 8 benchmarks tasks, SSL is actually *not* necessary for achieving high performance: thresholding and re-labeling can recover 89.1% of the gains enjoyed by also using SSL.
- To explain SSL's lack of impact, we find that the small amounts of unlabeled data in these benchmarks (i.e., $<$31%) are mostly unnecessary; when using clean instead of weak labels, ignoring unlabeled examples drops test accuracy by $<$2.5 p.p.
- In contrast, when WS sources leave more data as unlabeled (i.e., $>$65%), SSL is generally worth prioritizing; using SSL can improve upon thresholding and re-labeling by up to 16 p.p in such cases.

2. Related Work

SSL for WS tasks. Methods in WS have increasingly turned to SSL to improve end model training. This includes DENOISE (Ren et al., 2020), which incorporates the temporal ensembling SSL algorithm (Laine & Aila, 2017), as well as COSINE (Yu et al., 2021) and KeyClass (Gao et al., 2022), which both use self-training (Lee, 2013). Other works (Karamanolakis et al., 2021; Maheshwari et al., 2020; Chen et al., 2021; Mazzetto et al., 2021b;a; Pukdee et al., 2023; Awasthi et al., 2020) apply SSL when learning from weak labels *plus* a small set of clean labels. However, these methods fundamentally differ in their use of SSL, i.e., they define a labeled-unlabeled split based on whether an example has been cleanly or weakly labeled. Further, integrating clean labels into WS enables a greater variety of specialized strategies, so we do not consider this setting in our study. Likewise, Boecking & Dubrawski (2019) assume additional supervision via heuristics for which examples share the same labels, similar in spirit to consistency-based SSL methods. Finally, we defer a more thorough background on SSL in its own right to Section 3.

SSL for learning with noisy labels. SSL techniques have been regularly applied in the literature concerning learning from noisy labels (Li et al., 2020; Ding et al., 2018; Kong et al., 2019). Though this setting is similar to WS, its main difference is that label noise comes from a single "black-box" noising process instead of from multiple explicit WS sources. Thus, the resulting label noise patterns, often also artificially injected in input-independent ways (Wei et al., 2022), may differ significantly from those in WS. Furthermore, the WS setting can contain some examples with no labels since WS sources may abstain.

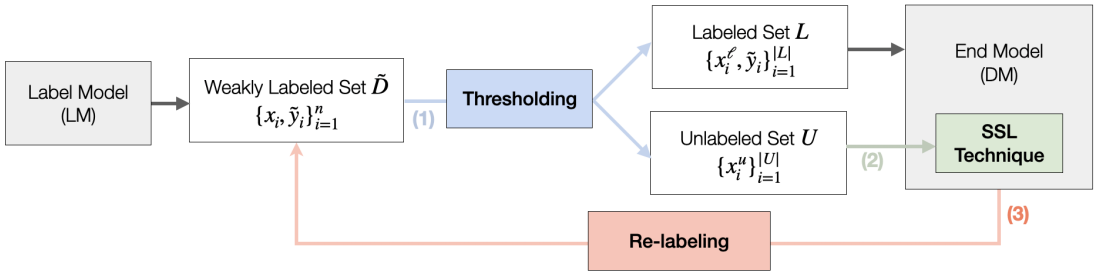
Subset selection in WS. Lang et al. (2022) showcases the broad utility of more carefully selecting subsets of weak labels before end model training. In our work, we consider subset selection in the greater context of two other trends in the WS literature, applying SSL and re-labeling. Compared to the core method of Lang et al. (2022), we also try a simpler baseline, similar to Gao et al. (2022), based on thresholding the existing confidences produced for weak labels. We find that this method offers a competitive alternative on most datasets.

3. Design Space

In this section, we first formalize WS and SSL. Then, we describe how our design space overlays on WS, detailing its three key axes along with the specific instantiations of each that we use in our experiments. Finally, we contextualize which parts of existing work fit within our framework.

3.1. Problem Formalization

Weak supervision. In WS, we start with an unlabeled training set $D = \{x_i\}_{i=1}^n \in \mathcal{X}^n$ drawn from an underlying distribution $(x, y) \sim P$. Labels are provided by a set of labeling functions (LFs) $\{\lambda_j\}_{j=1}^m$, where each $\lambda_j : \mathcal{X} \to \mathcal{Y} \cup \{\emptyset\}$ either labels or abstains (denoted as $\lambda_j(x_i) = \emptyset$)



Thresholding	{ Coverage, Confidence, Cut-based }
SSL Technique	{ None, EM, VAT, Self-Training, Self-Training (DM as LF) }
Re-labeling	{ None, DM directly, DM as LF }

Figure 1. Overview of our design space. In WS, a label model (LM) produces a weakly labeled training set $\tilde{D}$ which is used to train the discriminative end model (DM). Our design space overlays three decision points on this pipeline: (1) *thresholding*, which filters out some weak labels in $\tilde{D}$ to return labeled and unlabeled sets L and U; (2) *SSL technique*, which defines how the end model can still use U during training; (3) *re-labeling*, which uses the DM to update previously used weak labels.

on each x_i. The goal is to train a *discriminative end model* (DM) $f : \mathcal{X} \to \mathcal{Y}$ that performs well on P. Canonically, WS methods contain two components. First, a *label model* (LM) observes $\left(D, \{\lambda_j\}_{j=1}^m\right)$ and outputs a *weakly labeled* training set $\tilde{D} = \{(x_i, \tilde{y}_i)\}_{i=1}^n$. Second, f is trained using $\tilde{D}$. Often, the LM models $\tilde{y}_i$ probabilistically, allowing $\tilde{y}_i$ to be a soft-label, a vector of probabilities over $\mathcal{Y}$.

Problematically, $\tilde{D}$ may contain several *uncovered* examples, where $\lambda_j(x_i) = \emptyset$ for all λ_j. Thus, the default practice is to train f only on *covered* examples that received at least one non-abstaining LF vote. Concurrently, $\tilde{D}$ may also contain several *incorrect* labels, where $\tilde{y}_i \neq y_i$ (or in the soft-label case, $\arg\max_{c \in \mathcal{Y}} \tilde{y}_i[c] \neq y_i$ where $\tilde{y}_i[c]$ is the probability assigned to class c). Both issues may lead to sub-optimal performance compared to standard supervised learning. Here, we focus on whether SSL lets us more effectively learn from $\tilde{D}$ in light of these challenges.

Semi-supervised learning. SSL assumes access to a labeled dataset $L = \{(x_i^\ell, y_i)\}_{i=1}^{|L|}$ as well as an unlabeled dataset $U = \{x_i^u\}_{i=1}^{|U|}$. The goal is to obtain a model that performs well on L's underlying distribution despite the limited size of L. Though not strictly required, it is often assumed that $|L| << |U|$ and both are drawn from the same test distribution. Generally, each SSL method makes a core assumption about how $P(x)$ relates to $P(y|x)$. Popular categories of methods include *entropy-minimization* (Grandvalet & Bengio, 2004; Lee, 2013), which assumes $P(y|x)$ is uncertain only when $P(x)$ is small, and *consistency regularization* (Miyato et al., 2017; Tarvainen & Valpola, 2017; Laine & Aila, 2017), which assumes local smoothness of $P(y|x)$ when $x \sim P(x)$, encouraging the model to make similar predictions at similar inputs.

3.2. A Simple Design Space

By considering the standard WS pipeline, we anchor our design space on three natural decision points as shown in Figure 1: (1) thresholding strategy, (2) SSL technique, and (3) whether to re-label. Importantly, our design space is agnostic to the specification of LM and DM, though particular LMs and DMs could affect which methods work best.

Thresholding. To apply SSL, we must first define which examples in $\tilde{D}$ should be considered as part of L and U, respectively. Though this is part of the problem definition in traditional SSL settings, it is a non-trivial choice in WS. As a default, standard WS pipelines select L based on LF coverage, ignoring uncovered examples on which all LFs abstain. However, as shown by Lang et al. (2022), removing additional examples from $\tilde{D}$ can help if they are more likely to be incorrectly labeled. In our work, we consider the following strategies for partitioning $\tilde{D}$ into L and U:

- *Coverage-based (default):* This removes uncovered points $\{(x_i, \tilde{y}_i) : \lambda_j(x_i) = \emptyset, \forall j\}$.
- *Confidence-based:* Since most LMs can output probabilistic labels, a basic yet under-explored approach tried by Gao et al. (2022) is to remove examples that have low estimated confidence, here defined as the highest probability assigned to a class. Formally, this removes the examples $\{(x_i, \tilde{y}_i) : \max_{c \in \mathcal{Y}} \tilde{y}_i[c] < \xi\}$ for some threshold $|\mathcal{Y}|^{-1} < \xi < 1$.
- *Cut-based:* This is the method of Lang et al. (2022), which at a high-level removes examples whose labels differ most from those of their nearest neighbors in some pre-trained embedding space.

Method	Thresholding	SSL Technique	Re-labeling L	Novel LM	Add. Reg.	Add. Labels
Vanilla WS	Coverage	–	–	–	–	–
Cutstat (Lang et al., 2022)	Cut-based (one-time)	–	–	–	–	–
Denoise (Ren et al., 2020)	Coverage	Temp. Ensembling	Self-Train LM	✓	–	–
Weasel (Cachay et al., 2021)	–	–	Agreement-based	✓	–	–
Cosine (Yu et al., 2021)	Confidence (dyn.)	Self-Train	DM directly	–	✓	–
KeyClass (Gao et al., 2022)	Confidence (dyn.)	Self-Train	DM directly	–	✓	–
ASTRA (Karamanolakis et al., 2021)	Coverage	Self-Train (DM as LF)	DM as LF	✓	–	✓
LPA+WL (Pukdee et al., 2023)	–	Label Propagation	–	–	–	✓
SPEAR (Maheshwari et al., 2020)	–	Entropy Min	Agreement-based	✓	✓	✓

Table 1. Contextualizing methods within (middle columns) and outside (rightmost) our design space. "Novel LM" refers to a method introducing its own label model. "Add. Reg." refers to using additional regularizations, such as contrastive losses and soft-label re-normalization. "Add. Labels" refers to assuming an additional set of clean labels. For methods using clean labels, we do not compare to their results directly, but we can still place elements of their approaches in our design space.

SSL Techniques. After partitioning $\tilde{D}$ into L and U, we next consider how the end model can still make use of the examples in U. By default, WS methods ignore U altogether, but we may instead apply a variety of strategies, such as plugging in any existing SSL technique. Like Oliver et al. (2018), we limit ourselves to SSL techniques that add unsupervised loss terms during training since these methods tend to achieve the best performance on traditional SSL benchmarks. We also pick representative methods from their taxonomy of approaches:

- *Entropy minimization (EM):* A classical SSL approach, EM (Grandvalet & Bengio, 2004) penalizes less confident predictions on unlabeled examples, aiming for a decision boundary in low-density regions of $P(x)$.
- *Self-training (ST):* Self-training, along with the closely related pseudo-labeling (Lee, 2013; C. Rosenberg & Schneiderman, 2005), are traditional SSL methods that iteratively use current predictions on unlabeled examples as true labels. Uniquely in the WS setting, we also consider self-training the LM and DM *together*. Inspired by Karamanolakis et al. (2021), we specifically try feeding the end model as an additional LF λ_{m+1} used to re-fit the LM. We call this ST (DM as LF).
- *Consistency regularization:* These methods encourage models to make similar predictions on (realistic) perturbations of unlabeled examples. Of these methods, we use VAT (Miyato et al., 2017) as it does not rely on data augmentations, unlike most others. Augmentations are less straightforward to define for text datasets, which comprise the majority of WS benchmarks.

Re-labeling L. Traditional SSL assumes that all labels in L are correct. However, in WS, we may also consider using the end model to correct labels in L as it trains. This increasingly popular technique in WS methods (Cachay et al., 2021; Yu et al., 2021; Karamanolakis et al., 2021; Maheshwari et al., 2020) is often packaged with traditional self-training as an overall "SSL method" (Yu et al., 2021; Karamanolakis et al., 2021): instead of using the current model to label just U, these methods also do so for L. However, this makes it difficult to discern whether these methods improve performance because they leverage U (i.e., apply SSL) or simply because they clean up labels in L. Therefore, in our study, we explicitly decouple the use of SSL and re-labeling L as two *independent* decisions; i.e., we can re-label L *regardless* of whether we use any specific SSL technique.

Specifically, we consider two types of re-labeling: (1) using the end model's predictions directly as in ST, and (2) re-fitting the LM as in ST (DM as LF). For tractability, we treat re-labeling as a binary decision when accompanied by an SSL technique; we re-label L with the end model directly in except when the SSL technique is ST (DM as LF). Also, we formalize re-labeling as looping the two-stage WS pipeline back onto itself. In principle, N rounds of re-labeling can be paired with $N+1$ separate choices for thresholding and SSL. However, to reduce this search space, we consider either using the *same* thresholding and SSL techniques across all rounds or fixing the L/U split after thresholding the initial LM outputs; in the context of thresholding, we refer to these two schedules as *dynamic* and *one-time,* respectively. Finally, some recent methods re-label by using a specialized LM that can be jointly learned with the end model (Cachay et al., 2021; Maheshwari et al., 2020). We do not include this type of *agreement-based* re-labeling in our study, instead focusing on methods agnostic to the form of LM.

3.3. Contextualizing previous works

With our design space, we can contextualize several recent WS methods, as shown in Table 1. Overall, this table demonstrates the lack of systematic exploration. For instance, few works perform any thresholding beyond coverage-based, and only Lang et al. (2022) and Gao et al. (2022) threshold LM outputs. Furthermore, assessing the impact of SSL in WS settings is muddled because methods often incorporate several different techniques. A salient example is that COSINE (Yu et al., 2021) was found by WRENCH (Zhang et al., 2021) to obtain state-of-the-art performance, with both works championing the usage of unlabeled data as a key driver of improvements; however, what COSINE refers to

Datasets	$\lvert\mathcal{Y}\rvert$	|Train|	|Test|	Coverage	Snork. Precision
IMDb	2	20000	2500	87.6%	74.4%
Yelp	2	30400	3800	82.8%	75.4%
Youtube	2	1586	250	87.7%	87.0%
AgNews	4	96000	12000	69.1%	82.5%
Trec	6	4965	500	95.1%	60.0%
Spouse	2	22254	2701	25.8%	65.6% (on Val)
Chemprot	10	12861	1607	85.6%	58.0%
Census	2	10083	16281	99.1%	58.0%

Table 2. Statistics for the data/LF sets that we use, groupings are by task type: text, text relation, and tabular classification. Note that *Spouse* does not come with ground-truth training labels.

as self-training actually involves pseudo-labeling the *whole* dataset, thereby performing *both* SSL and re-labeling. Further, many methods use techniques outside of our design space entirely, such as novel LMs (Ren et al., 2020; Cachay et al., 2021; Karamanolakis et al., 2021; Maheshwari et al., 2020), contrastive learning (Yu et al., 2021), and soft-label re-normalization (Yu et al., 2021; Gao et al., 2022). As a result, it remains unclear whether our three axes are necessary or even sufficient to achieve optimal performance. Though our design space is by no means exhaustive, we believe it offers a useful starting point to answer such questions.

4. Results

We begin by describing our experiment setup and various baselines in Section 4.1. Then, Section 4.2 explores how our design space yields methods that perform at least as well as the aforementioned baselines. In Sections 4.3 and 4.4, we conduct extensive ablations on our three axes, finding that SSL is surprisingly unnecessary on most WS benchmarks. Finally, Section 4.5 explains this phenomenon and explores settings in which SSL is more helpful.

4.1. Experimental Setup

Datasets and models. We use 8 classification datasets (see Table 2) and largely follow the end model configurations from WRENCH (Zhang et al., 2021) with a few changes to the hyperparameter grid (Appendix A). For NLP tasks, we both fine-tune RoBERTa pre-trained models and train MLP classification heads on (frozen) RoBERTa embeddings, deferring results for the latter to the appendix. For tabular tasks, we just train MLPs. We tune all methods on a shared hyperparameter budget of 300 trials for MLPs and 50 trials for full RoBERTa fine-tuning. All reported test performances are then averages over over three additional runs, while all error bars are the standard deviations over these runs. Finally, though our design space is compatible with any LM, we use the soft-labels produced by the Snorkel LM from Ratner et al. (2019). However, we test robustness to this choice by also trying Majority Voting when comparing with existing methods. Zhang et al. (2021) found these two LMs to be the most consistent across many WS tasks.

Baselines. We consider five baselines, corresponding to the WRENCH implementations of existing state-of-the-art methods. As Table 1 shows, each incorporates some aspects of our design space, while not necessarily being contained completely within it. We list these methods as follows:

Vanilla: applies the default WS pipeline, using coverage thresholding and no SSL or re-labeling.

Cutstat + {*Snorkel, MV*}: uses the thresholding method from Lang et al. (2022), with no alterations.

COSINE + {*Snorkel, MV*}: modifies the method from Yu et al. (2021) in two ways when appropriate for fair comparison; the LM used is Snorkel instead of MV except as shown in Table 8, and the DM is an MLP in Table 6.

Denoise + {*Snorkel, MV*}: applies the method from Ren et al. (2020) except using Snorkel instead of MV to initialize their label aggregator as shown in Table 8. Also, we use RoBERTa-based end models instead of BERT.

Weasel: applies the method from Cachay et al. (2021) except to train RoBERTa-based end models on the text datasets.

4.2. Does our design space yield competitive methods?

We first demonstrate that simple methods from our design space can at least match the performance of all previous methods. When the end model is RoBERTa, our best *single method* (shown in Figure 2) is the combination of (dynamic) confidence thresholding, ST (DM as LF), and re-labeling. This method at least matches the performance of other baselines on 5 of the 7 text-based datasets, on average closing 11.5% of the remaining gap between previous WS methods and fully supervised learning. On *AGNews* and *Chemprot*, only *Cutstat* results in a better point estimate. However, for both tasks, swapping cut-based thresholding into our method produces state-of-the-art accuracies of 0.885 (0.002) and 0.601 (0.008), respectively (as shown in Appendix B, this method is similarly strong when the LM is instead Majority Voting). Finally, Figure 2 shows that searching our design space exhaustively *per dataset*–while not necessarily practical–can be even more effective, closing an additional 15% of the gap to fully supervised learning. Significantly, our methods are strong despite not using some additional techniques employed by recent works (i.e., see Table 1). This provides empirical justification for focusing only on methods from within our design space in later analyses.

4.3. Which axes are most important?

Though our design space is *sufficient* for strong performance, we now show that SSL is, by and large, not necessary on current benchmarks. Specifically, we conduct an extensive ablation comparing all eight possible subsets of including (or ignoring) each axis, presenting the results in Tables 3

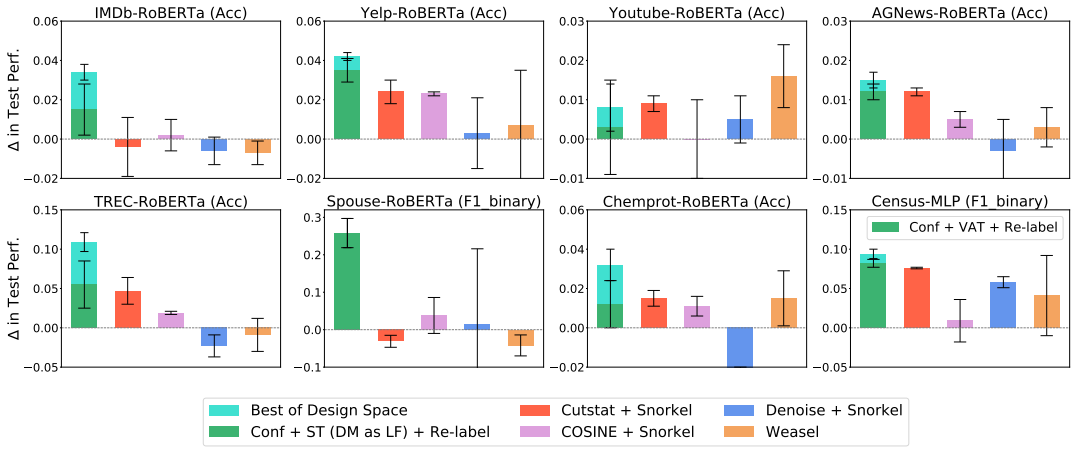


Figure 2. Test performance for LM = Snorkel. We compare the best *single* method found for each DM from our design space (green) with the best methods *per-dataset* (turquoise, narrower error caps) as well as four recent proposals from the literature. We plot all performances relative to that of vanilla training (deferring absolute metrics to Table 7 in Appendix B). Note that for Census, a tabular dataset, the best single method is different since we report the one we found for MLPs (see Appendix B).

and 9. As an example, for the row "Thresh + SSL," we run every combination within the cross-product of all *non-default* thresholding and SSL techniques (i.e., {Conf-based, Cut-based} × {EM, VAT, ST, ST w/ DM}) and then select the best of these combinations using the validation set. For "Entire Design Space," we perform this method selection procedure over all methods from the design space.[1]

From this analysis, we observe that SSL helps only in limited scenarios. Including SSL on *Spouse* yields a 0.2 increase in F1-score compared to not using any form of SSL. However, on all the other six tasks, "Thresh + Re-label" performs at least within a standard deviation of the best method, making up 89.1% of the gap between "Vanilla" and "Thresh + SSL + Re-label." One explanation for this result is the distinctly lower coverage LF set for *Spouse*, a factor we explore in depth in Sec 4.5. Further, model size may also play a role. In the corresponding Table 9 (Appendix B) for MLPs, "Thresh + Re-label" can make up only 68.2% of the gap between "Vanilla" and "Thresh + SSL + Re-label."

4.4. What are the best instantiations of each axis?

Having compared the three axes at a macro-level, we now zoom in on each. Specifically, we compare all implementations of a given axis when paired with the best possible setting of the other two. For thresholding and SSL, we find that previously underexplored approaches are worth using.

Thresholding. Examining Table 4, we observe that thresholding is significantly helpful in most cases. This extends the conclusions of Lang et al. (2022), showing that removing labels is still adds unique value *even when also allowing for SSL and re-labeling*. Interestingly, the simpler confidence-based threshold matches the cut-based method (within error bars) on all datasets except *Chemprot*.

SSL and Re-labeling. For these two axes, we observe largely similar trends in Tables 10 and 11 in Appendix C.2. SSL and re-labeling are each only strictly necessary for a minority of datasets, significantly outperforming "no SSL" and "no re-labeling" on just one and two datasets, respectively. As such, all SSL methods tend to perform similarly. But for MLPs in Table 13, SSL is more useful and we see that VAT and ST (DM as LF) are the most consistent. No other method comes within error bars on four tasks.

4.5. When is SSL more useful?

To explain why SSL is largely redundant on most existing WS benchmarks, we show that the unlabeled data in each task is generally unnecessary for learning strong models. However, when using lower coverage LF sets (a setting not captured by these benchmarks), ignoring unlabeled data can significantly compromise performance. In these settings, SSL has more room to be impactful.

Data gaps in WS. We can think of any WS training set $L = \{(X_i^\ell, \tilde{y}_i)\}_{i=1}^{|L|} \subseteq \tilde{D}$ as suffering from three deficiencies:

1. *Limited size:* Since not all examples are labeled, the quantity of labels may be insufficient.
2. *Coverage bias:* Since LFs abstain based on feature-dependent rules, the inputs in L are a biased subpopulation of the full test distribution.
3. *Label noise:* Since LFs are just heuristics, labels $\tilde{y}_i$ can be biased towards incorrect classes.

[1]This differs from "Thresh + SSL + Re-label" because methods for that row must employ *non-default* choices for each axis.

Method	IMDb	Yelp	Youtube	AGNews	TREC	Spouse	Chemprot	Mean	w/o Spouse
Vanilla	0.873 (0.001)	0.919 (0.009)	**0.944 (0.007)**	0.870 (0.003)	0.637 (0.010)	0.273 (0.074)	0.569 (0.001)	0.726 (0.011)	0.802 (0.003)
Thresh Alone	0.887 (0.008)	0.948 (0.006)	**0.953 (0.002)**	**0.882 (0.001)**	0.703 (0.017)	0.267 (0.060)	0.584 (0.004)	0.746 (0.009)	0.826 (0.003)
SSL Alone	0.885 (0.002)	0.939 (0.008)	**0.955 (0.005)**	**0.881 (0.003)**	0.640 (0.013)	0.343 (0.075)	**0.593 (0.010)**	0.748 (0.011)	0.816 (0.003)
Re-label Alone	**0.901 (0.013)**	0.944 (0.004)	**0.952 (0.006)**	0.877 (0.002)	0.700 (0.014)	0.332 (0.105)	0.577 (0.010)	0.755 (0.015)	0.825 (0.004)
Thresh + SSL	0.885 (0.018)	**0.959 (0.002)**	**0.943 (0.008)**	**0.887 (0.005)**	0.732 (0.014)	**0.529 (0.057)**	**0.592 (0.005)**	0.790 (0.009)	0.833 (0.004)
Thresh + Re-label	**0.905 (0.011)**	**0.961 (0.002)**	**0.952 (0.006)**	**0.883 (0.005)**	**0.748 (0.023)**	0.236 (0.085)	**0.600 (0.006)**	0.755 (0.013)	0.841 (0.005)
SSL + Re-label	0.894 (0.008)	0.949 (0.002)	**0.951 (0.002)**	0.879 (0.002)	0.702 (0.035)	0.365 (0.073)	0.583 (0.003)	0.760 (0.012)	0.826 (0.006)
Thresh + SSL + Re-label	**0.907 (0.004)**	**0.961 (0.003)**	**0.949 (0.010)**	**0.885 (0.003)**	**0.765 (0.012)**	**0.531 (0.039)**	**0.601 (0.008)**	0.800 (0.006)	0.845 (0.003)
Entire Design Space	**0.907 (0.004)**	**0.961 (0.002)**	**0.952 (0.006)**	**0.887 (0.005)**	**0.765 (0.012)**	**0.531 (0.039)**	**0.601 (0.008)**	0.801 (0.006)	0.845 (0.003)
Fully supervised	0.932 (0.005)	0.976 (0.001)	0.967 (0.013)	0.918 (0.006)	0.966 (0.004)	– –	0.894 (0.012)	– –	0.943 (0.003)

Table 3. Axis ablation for LM = Snorkel, DM = RoBERTa. Each row corresponds to picking a specific method within our design space using validation tuning. Blue numbers are the highest for a given dataset, while bold numbers are those within error bars of the best result.

Thresh	IMDb	Yelp	AGNews	TREC	Spouse	Chem.
Cov	**0.901 (0.013)**	0.949 (0.002)	**0.881 (0.003)**	0.702 (0.035)	0.365 (0.073)	0.593 (0.010)
Cut	**0.907 (0.004)**	**0.958 (0.003)**	**0.887 (0.005)**	**0.765 (0.012)**	**0.531 (0.007)**	**0.608 (0.003)**
Conf	**0.906 (0.005)**	**0.961 (0.003)**	**0.884 (0.002)**	**0.753 (0.021)**	**0.531 (0.039)**	0.593 (0.010)

Table 4. Deeper dive into the thresholding axis for DM = RoBERTa. We report the highest performance of a method that incorporates a given type of thresholding, selected by validation performance.

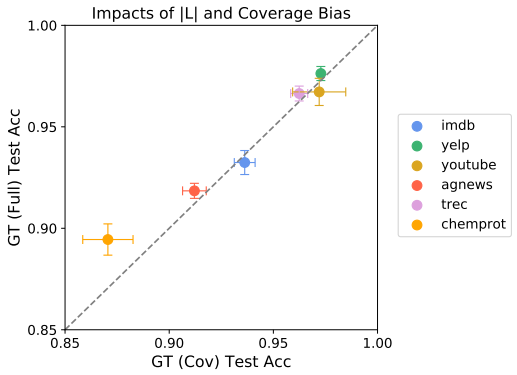


Figure 3. Measuring the impact of smaller $|L|$ and coverage bias on WS benchmarks. As seen, when label noise is removed from the covered set of examples for *GT (Cov)*, the performance drops <2% compared to having all the clean labels for *GT (Full)*. Note we cannot plot *Spouse* as it does not have clean training labels.

We hypothesize that SSL adds more unique value within our design space when gaps (1) and (2) are more significant. When gap (3) is the only dominating factor, there is less reason to expect SSL to outperform thresholding and re-labeling since the latter two more directly address label noise. However, these two axes still do not use the unlabeled examples, which cause gaps (1) and (2).

Label noise is the main gap on WS benchmarks. To measure the relative importance of the three data gaps, we compare the following models:

GT (Cov): the model trained on the clean labels for only covered inputs, removing gap (3) but retaining (1) and (2).

GT (Full): the model trained with a clean and fully labeled version of D, removing all gaps.

Given our hypothesis, we would expect these models to perform *similarly* on WRENCH benchmarks. Indeed, as shown in Figure 3, the largest gap between them is <2 p.p.

SSL helps more when coverage is lower. While SSL's ineffectiveness on existing benchmarks corresponds to label noise being the predominant data gap, we would ideally also show that SSL is *more* useful when the other two gaps are significant, i.e., when *GT (Cov)* performs markedly worse than *GT (Full)*. One natural way to explore this would be to test on *lower coverage* LF sets, which result in more unlabeled data. However, most existing benchmarks have coverages above 80% (see Table 2). To overcome this limitation, we first explore a wider range of coverages by subsampling or generating LFs on existing datasets. We also create two new WS text classification tasks based on publicly available datasets, *Massive18* (FitzGerald et al., 2022) and *Banking77* (Casanueva et al., 2020); these tasks have larger label spaces than all WRENCH datasets and thus require more effort (i.e., LFs) to obtain high coverage. Finally, we explore the less-studied tabular setting, using *Mushroom*, *Spambase*, and

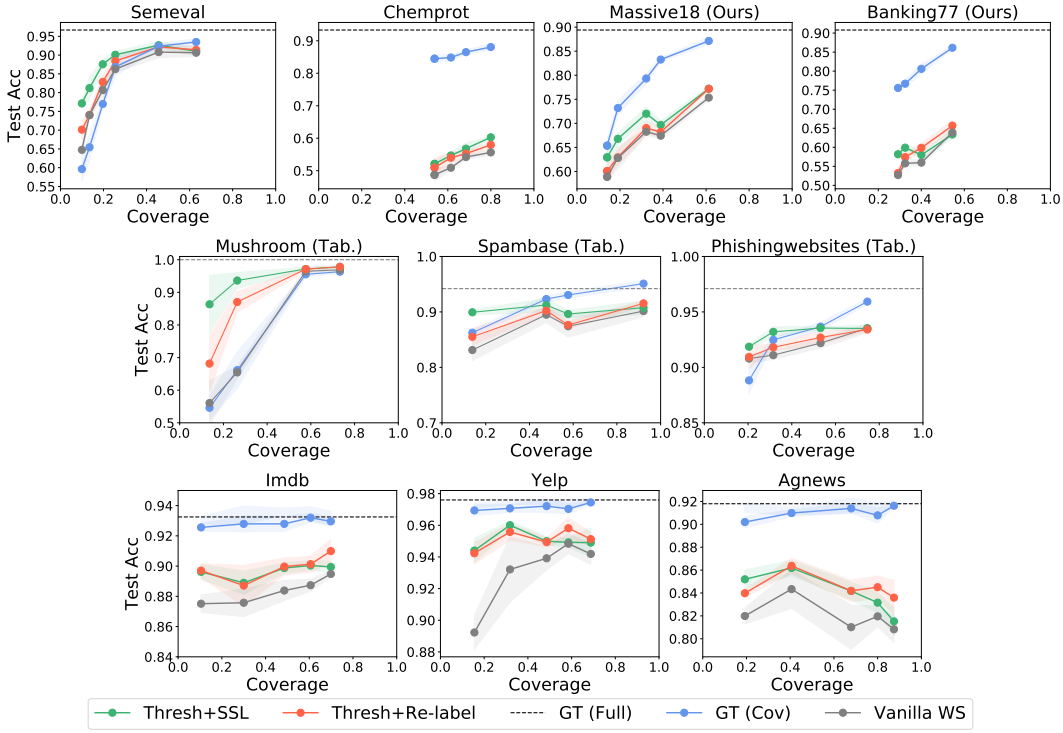


Figure 4. Impacts of coverage level on data gaps and the effectiveness of SSL. Plotting *GT (Full)* (gray-dashed) and *GT (Cov)* (blue), we measure the impacts of removing label noise across nested LF subsets. We also compare the effectiveness of Thresh + SSL (green) to Thresh + Re-label (red). For datasets more impacted by coverage bias and limited size (larger gaps between dashed and blue lines), SSL adds more unique value. "(Ours)" refers to datsets we introduce. "(Tab.)" refers to tabular tasks.

PhishingWebsites in a setup similar to that of Zhang et al. (2022b) (see Appendix D for details on all these datasets).

From this analysis, we first see that smaller coverage levels (within the ranges we test) do not always cause *GT (Cov)* to perform poorly. On some tasks (top two rows of Figure 4), including the three tabular datasets and our two new ones, *GT (Cov)* performs significantly worse as coverage decreases.[2] In contrast, on *IMDb*, *Yelp*, and *AGNews* (bottom row of Figure 4), *GT (Cov)* surprisingly comes within 2% of *GT (Full)* even when coverage drops to 10-20%.

Importantly, the partitioning of datasets based on the performance of *GT (Cov)* also corresponds to the effectiveness of using SSL over not using it. For each LF set used, we run the methods within a reduced version of our design space: we allow for confidence thresholding and try both versions of self-training to perform SSL (i.e., by labeling points in U) or to re-label (i.e., by labeling points in L). For the tabular tasks, we also try VAT for the SSL technique because of its effectiveness for MLPs. We plot the best performing methods for both "Thresh + SSL" and "Thresh + Re-label" in Figure 4. As shown in the top two rows, just "Thresh + SSL" is enough to consistently outperform "Thresh + Re-label" at lower coverage levels. In contrast, in the bottom row, where *GT (Cov)* drops in performance by $<$2 p.p., "Thresh + SSL" continues to perform within errors of "Thresh + Re-label."

Overall, this suggests that SSL can indeed be useful in WS settings that differ from those captured by standard benchmarks. At lower coverages (i.e., $<$35%), SSL is consistently worth trying; potentially allowing users to reduce the number of LFs they need to write in order to achieve a particular target performance. On our tabular tasks, SSL even allows one to match having 40 LFs (highest coverage plotted) with at most 20 LFs (second lowest).

5. Conclusions

We proposed a design space for combining SSL and WS, using it to contextualize and match the performance of state-of-the-art WS methods. We show that on existing WS benchmarks, using the unlabeled data is surprisingly not essential. However, it can be more useful when training MLPs instead of RoBERTa and when the LFs cover fewer examples. Some future directions include developing: (1) heuristics for more efficiently navigating our design space given a new task; (2) ways to compare data gaps without using clean labels.

[2] In Appendix E, we show that coverage bias (and not limited size) is primarily responsible for these performance drops.

References

Awasthi, A., Ghosh, S., Goyal, R., and Sarawagi, S. Learning from rules generalizing labeled exemplars. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkeuexBtDr.

Boecking, B. and Dubrawski, A. Pairwise feedback for data programming. In *Proceedings of NeurIPS '19 Workshop on Learning with Rich Experience (LIRE '19)*, December 2019.

C. Rosenberg, M. H. and Schneiderman, H. Semi-supervised learning by entropy minimization. In *Semi-Supervised Self-Training of Object Detection Models*, 2005.

Cachay, S., Boecking, B., and Dubrawski, A. End-to-end weak supervision. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/0e674a918ebca3f78bfe02e2f387689d-Paper.pdf.

Casanueva, I., Temčinas, T., Gerz, D., Henderson, M., and Vulić, I. Efficient intent detection with dual sentence encoders. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pp. 38–45, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.nlp4convai-1.5. URL https://aclanthology.org/2020.nlp4convai-1.5.

Chen, M., Cohen-Wang, B., Mussmann, S., Sala, F., and Re, C. Comparing the value of labeled and unlabeled data in method-of-moments latent variable estimation. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 3286–3294. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/chen21g.html.

Ding, Y., Wang, L., Fan, D., and Gong, B. A semi-supervised two-stage approach to learning from noisy labels. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1215–1224, Los Alamitos, CA, USA, mar 2018. IEEE Computer Society. doi: 10.1109/WACV.2018.00138. URL https://doi.ieeecomputersociety.org/10.1109/WACV.2018.00138.

FitzGerald, J., Hench, C., Peris, C., Mackie, S., Rottmann, K., Sanchez, A., Nash, A., Urbach, L., Kakarala, V., Singh, R., Ranganath, S., Crist, L., Britan, M., Leeuwis, W., Tur, G., and Natarajan, P. Massive: A 1m-example multilingual natural language understanding dataset with 51 typologically-diverse languages, 2022. URL https://doi.org/10.48550/arXiv.2204.08582.

Gao, C., Goswami, M., Chen, J., and Dubrawski, A. Classifying unstructured clinical notes via automatic weak supervision. In *Proceedings of the 7th Machine Learning for Healthcare Conference*, volume 182 of *Proceedings of Machine Learning Research*, pp. 673–690. PMLR, 05–06 Aug 2022. URL https://proceedings.mlr.press/v182/gao22a.html.

Grandvalet, Y. and Bengio, Y. Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2004. URL https://proceedings.neurips.cc/paper/2004/file/96f2b50b5d3613adf9c27049b2a888c7-Paper.pdf.

Karamanolakis, G., Mukherjee, S., Zheng, G., and Awadallah, A. H. Self-training with weak supervision. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 845–863, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.66. URL https://aclanthology.org/2021.naacl-main.66.

Kocoń, J., Cichecki, I., Kaszyca, O., Kochanek, M., Szydło, D., Baran, J., Bielaniewicz, J., Gruza, M., Janz, A., Kanclerz, K., Kocoń, A., Koptyra, B., Mieleszczenko-Kowszewicz, W., Miłkowski, P., Oleksy, M., Piasecki, M., Radliński, , Wojtasik, K., Woźniak, S., and Kazienko, P. ChatGPT: Jack of all trades, master of none. *Information Fusion*, 99:101861, 2023. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2023.101861. URL https://www.sciencedirect.com/science/article/pii/S156625352300177X.

Kong, K., Lee, J., Kwak, Y., Kang, M., Kim, S. G., and Song, W.-J. Recycling: Semi-supervised learning with noisy labels in deep neural networks. *IEEE Access*, 7:66998–67005, 2019. doi: 10.1109/ACCESS.2019.2918794. URL https://ieeexplore.ieee.org/document/8721656.

Laine, S. and Aila, T. Temporal ensembling for semi-supervised learning. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=BJ6oOfqge.

Lang, H., Vijayaraghavan, A., and Sontag, D. Training subset selection for weak supervision. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16023–16036. Curran Associates, Inc.,

2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/66720ca4e5a09ff83b55a117a6b2a86c-Paper-Conference.pdf.

Lee, D. The simple and efficient semi-supervised learning method for deep neural networks. In *ICML Workshop on Challenges in Representation Learning*, 2013.

Li, J., Socher, R., and Hoi, S. C. Dividemix: Learning with noisy labels as semi-supervised learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJgExaVtwr.

Maheshwari, A., Chatterjee, O., Killamsetty, K., Iyer, R. K., and Ramakrishnan, G. Data programming using semi-supervision and subset selection. *CoRR*, abs/2008.09887, 2020. URL https://arxiv.org/abs/2008.09887.

Mazzetto, A., Cousins, C., Sam, D., Bach, S. H., and Upfal, E. Adversarial multi class learning under weak supervision with performance guarantees. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7534–7543. PMLR, 18–24 Jul 2021a. URL https://proceedings.mlr.press/v139/mazzetto21a.html.

Mazzetto, A., Sam, D., Park, A., Upfal, E., and Bach, S. Semi-supervised aggregation of dependent weak supervision sources with performance guarantees. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 3196–3204. PMLR, 13–15 Apr 2021b. URL https://proceedings.mlr.press/v130/mazzetto21a.html.

Miyato, T., Maeda, S.-i., Koyama, M., and Ishii, S. Virtual adversarial training: A regularization method for supervised and semi-supervised learning, 2017. URL https://arxiv.org/abs/1704.03976.

Oliver, A., Odena, A., Raffel, C. A., Cubuk, E. D., and Goodfellow, I. Realistic evaluation of deep semi-supervised learning algorithms. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://proceedings.neurips.cc/paper/2018/file/c1fea270c48e8079d8ddf7d06d26ab52-Paper.pdf.

Pukdee, R., Sam, D., Ravikumar, P. K., and Balcan, N. Label propagation with weak supervision. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=aCuFa-RRqtI.

Ratner, A., Hancock, B., Dunnmon, J., Sala, F., Pandey, S., and Ré, C. Training complex models with multi-task weak supervision. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01): 4763–4771, Jul. 2019. doi: 10.1609/aaai.v33i01.33014763. URL https://ojs.aaai.org/index.php/AAAI/article/view/4403.

Ratner, A. J., De Sa, C. M., Wu, S., Selsam, D., and Ré, C. Data programming: Creating large training sets, quickly. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/6709e8d64a5f47269ed5cea9f625f7ab-Paper.pdf.

Ren, W., Li, Y., Su, H., Kartchner, D., Mitchell, C., and Zhang, C. Denoising multi-source weak supervision for neural text classification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 3739–3754, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.334. URL https://aclanthology.org/2020.findings-emnlp.334.

Tarvainen, A. and Valpola, H. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/68053af2923e00204c3ca7c6a3150cf7-Paper.pdf.

Wei, J., Zhu, Z., Cheng, H., Liu, T., Niu, G., and Liu, Y. Learning with noisy labels revisited: A study using real-world human annotations. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=TBWA6PLJZQm.

Yu, Y., Zuo, S., Jiang, H., Ren, W., Zhao, T., and Zhang, C. Fine-tuning pre-trained language model with weak supervision: A contrastive-regularized self-training approach. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1063–1077, 2021. URL https://aclanthology.org/2021.naacl-main.84.pdf.

Zhang, J., Yu, Y., Li, Y., Wang, Y., Yang, Y., Yang, M., and Ratner, A. WRENCH: A comprehensive benchmark for weak supervision. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Bench-*

marks Track, 2021. URL https://openreview.net/forum?id=Q9SKS5k8io.

Zhang, J., Hsieh, C.-Y., Yu, Y., Zhang, C., and Ratner, A. A survey on programmatic weak supervision. *arXiv preprint arXiv:2202.05433*, 2022a. URL https://doi.org/10.48550/arXiv.2202.05433.

Zhang, J., Wang, H., Hsieh, C.-Y., and Ratner, A. J. Understanding programmatic weak supervision via source-aware influence function. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 2862–2875. Curran Associates, Inc., 2022b. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/1343edb2739a61a6e20bd8764e814b50-Paper-Conference.pdf.

Zhou, W., Lin, H., Lin, B. Y., Wang, Z., Du, J., Neves, L., and Ren, X. Nero: A neural rule grounding framework for label-efficient relation extraction. In *Proceedings of The Web Conference 2020*, WWW '20, pp. 2166–2176. Association for Computing Machinery, 2020. ISBN 9781450370233. doi: 10.1145/3366423.3380282. URL https://doi.org/10.1145/3366423.3380282.

Zhu, Y., Zhang, P., Haq, E.-U., Hui, P., and Tyson, G. Can chatgpt reproduce human-generated labels? a study of social computing tasks, 2023. URL https://doi.org/10.48550/arXiv.2304.10145.

A. Experiment Details

A.1. Search Spaces

Method	Hyperparameters	Description	Range
Snorkel LM	`lr`	learning rate	1e-5,1e-4,1e-3,1e-2,1e-1
	`weight_decay`	weight decay	1e-5,1e-4,1e-3,1e-2,1e-1
	`num_epoch`	the number of training epochs	5,10,50,100,200
MLP	`batch_size`	batch size	32,128,512
	`lr`	learning rate	1e-4,1e-3,1e-2
	`dropout`	dropout probability	0.2, 0.0
	`weight_decay`	weight decay	0.0
	`num_layer`	the number of hidden layers	2
	`hidden_size`	the hidden size of MLP layers	128, 256, 512
BERT	`batch_size`	batch size	32
	`lr`	learning rate	1e-5,3e-5,5e-5
	`weight_decay`	weight decay	1e-4
COSINE	T	period for updating pseudo-labels	50,100,200
	ξ	confidence threshold	0.1, 0.3, 0.5, 0.7, 0.9
	λ	weight for confidence regularization	0.01,0.05,0.1
	μ	weight for contrastive regularization	1
	γ	margin for contrastive regularization	1
Denoise	α	momentum term for temporal ensembling	0.6
	`d`	size of hidden layer	$2^6 - 2^9$
	`c1`	coefficient of denoiser loss	0.1,0.3,0.5,0.7,0.9
	`c2`	coefficient of classifier loss	0.1,0.3,0.5,0.7,0.9
	`c3`	coefficient of unsupervised self-training loss	1-`c2`-`c1`
WeaSEL	γ	temperature	1.0, 0.33
	`d`	size of hidden layer	$2^6 - 2^9$
	`dropout`	dropout prob	0.3
Confidence Thresh	`threshold`	minimum confidence to keep	`np.linspace`($\frac{1}{\lvert\mathcal{Y}\rvert}$ + 0.1, 0.9, 9)
Cutstat Thresh	`percentile`	what percentile as cut-off	`np.linspace`(0.1, 0.9, 9)
Self-Training	T	period for updating pseudo-labels	50, 100
Self-Train (DM as LF)	T	period for updating pseudo-labels	50, 100
EM	λ	the coefficient for EM loss	1, 0.5, 1e-1, 1e-2, 1e-3
VAT	λ	the coefficient for VAT unsupervised loss	1, 0.5, 1e-1, 1e-2, 1e-3
	`VAT ip`	Iterations of the power method for VAT	1
	ξ	Finite difference for approximation in VAT	0.05, 0.1, 0.5, 1, 5
	ϵ	VAT Perturbation distance	1e-6, 1e-3, 1, 2.5, 5

Table 5. Search spaces organized by type of method: label models, end models, baselines, thresholding, and SSL techniques.

A.2. Implementation Details

Label Models. Of the two label models that we use, we must tune parameters only for Snorkel. We tune over the search space in Table 5 once per dataset; we then fix the hyperparameters and seed for fitting Snorkel to ensure the same exact weak labels are given to all methods. This seed is chosen from a pool of three, also based on the validation set.

End Models. For MLPs, we increase the range of hidden sizes compared to WRENCH and tune for dropout instead of weight decay. For RoBERTa, we do not tune the batch size in order to reduce the search space, observing minimal differences compared to using an alternative batch size of 16. Following WRENCH, we perform early stopping based on validation performance for all methods. Specifically, we use a patience of 1000 steps for MLPs and 100 steps for RoBERTa.

Thresholding. Whenever applying *dynamic* versions of thresholding, we use the same threshold value across different rounds for simplicity. Setting separate thresholds for different stages of learning (or adaptively) could be an interesting direction for future work.

SSL Techniques. We make note of some important details relevant to specific SSL techniques:

- For the two self-training methods, we use a training schedule similar to that of COSINE, whereby we divide training into two stages: In the first stage, the end model is trained on weak labels as is done in standard WS end model training. In the second stage, new pseudo-labels are generated upon reaching a pre-specified update period of $\{50,100\}$ steps.
- For self-training, we also allow for applying thresholds to the pseudo-labels given to unlabeled data. When combining this approach with thresholding the initial weak labels, we again use the same threshold value.
- When using ST (DM as LF), we do not re-tune the label model-specific hyperparameters for Snorkel in subsequent self-training rounds. We default to instead using the same hyperparameters found from the initial search.
- For EM and VAT, we sample batches with equal numbers of labeled and unlabeled examples when taking each step but tune a weighting parameter to balance their losses.
- When applying VAT to RoBERTa models, we calculate perturbations with respect to the RoBERTa-based features (which are continuous) instead of the raw text inputs.

Re-labeling. For re-labeling, we again use a two-stage training schedule where the second stage contains a label update period (as explained when discussing self-training). When combining re-labeling with thresholding, we consider the set of examples with candidate labels (i.e., which are possibly removed when performing dynamic thresholding) to be fixed after the initial round of thresholding (of LM outputs). Thus, re-labeling provides new pseudo-labels only for examples that survived the initial thresholding.

Compute. We ran all MLP experiments on AWS, using up to four `g4dn.4xlarge` EC2 instances at one time. Each instance allowed for running up to 10 different experiments (i.e., here considered as a hyperparameter sweep for any specific method from our design space) in parallel. For all RoBERTa training runs, we ran our experiments on a fleet of up to 15 NVIDIA-A40 GPUs hosted on a cluster shared by our research lab. Each experiment fits on a single A-40 GPU, which is large enough to use our batch sizes without needing gradient accumulation.

B. Additional Results for Baseline Comparisons

B.1. Additional Plots

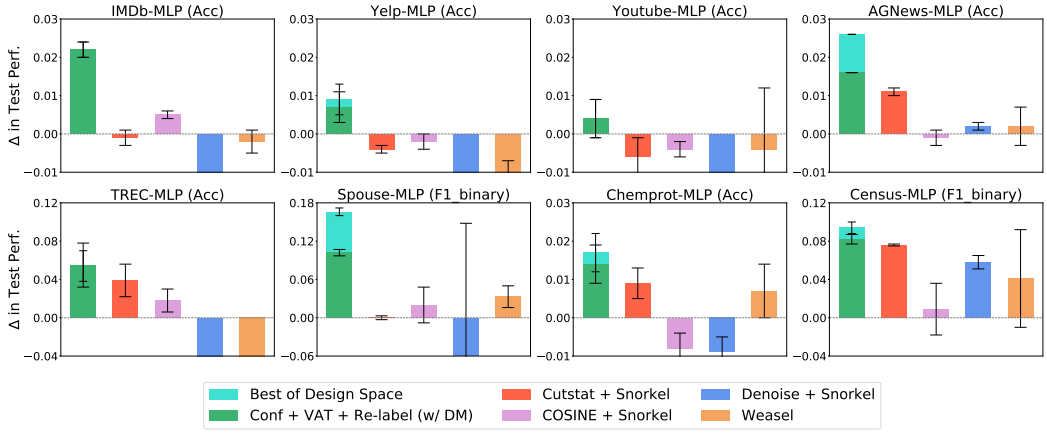


Figure 5. Test performance for LM = Snorkel, DM = MLP. We compare a selected method from our design space (green) with four recent proposals from the literature. We plot all performances relative to that of vanilla training (see Table 6 for the absolute metrics)

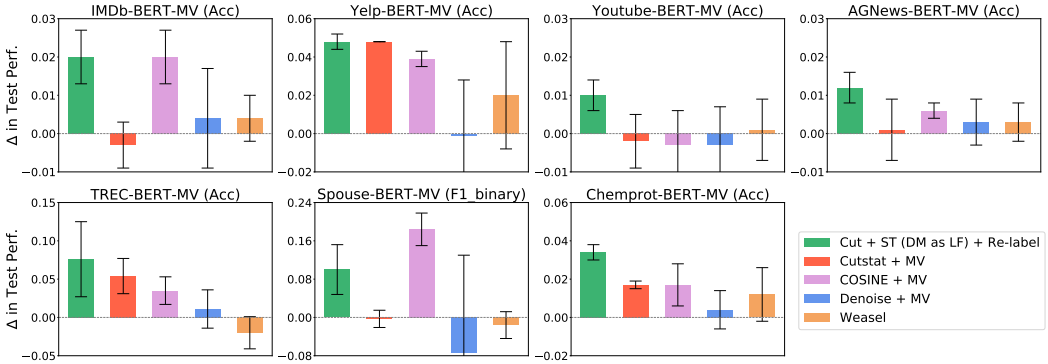


Figure 6. Test performance for LM = MV, DM = RoBERTa. We compare a selected method from our design space (green) with four recent proposals from the literature. We plot all performances relative to that of vanilla training (see Table 8 for the absolute metrics). For this setting, we did not run an exhaustive search over the whole design space so we do not report "Best of Design Space."

MLP Results. While fully fine-tuning RoBERTa end models provides better overall performance across text-based tasks, we also provide results for MLP end models for completeness. Here, the best single-method across datasets was (one-time) confidence thresholding + VAT + re-labeling. This method achieves the best point estimate on all 8 datasets compared to previous baselines.

Majority Voting Results. We also ablate the specific choice of label model to be Majority Voting for the RoBERTa experiments. Here, we mostly use the same method as for when the label model is Snorkel. However, we found that swapping in cut-based thresholding tended to perform better than sticking with confidence-based. This behavior can perhaps be explained by the relatively cruder confidence estimates for Majority Voting soft-labels; instead of learning a probabilistic model, the soft-labels for Majority Voting simply use the ratios of LF votes (e.g. for a binary task, if an example received 2 negative and 3 positive votes, the soft-label would be [0.4, 0.6]). Overall, we find that this method can least match previous baselines on all datasets except *Spouse*. Notably though, the best overall performance on *Spouse* is still obtained by applying our design space on top of the Snorkel label model (see Tables 7 and 8).

B.2. Tables with Absolute Performance

Method	IMDb	Yelp	Youtube	AGNews	TREC	Spouse	Chemprot	Census
Vanilla	0.828 (0.001)	0.917 (0.001)	**0.925 (0.004)**	0.854 (0.002)	0.623 (0.004)	0.228 (0.013)	0.544 (0.002)	0.511 (0.012)
Cutstat + Snorkel	0.827 (0.002)	0.913 (0.001)	**0.919 (0.005)**	0.865 (0.001)	**0.662 (0.017)**	0.228 (0.003)	**0.553 (0.004)**	0.587 (0.001)
Conf + VAT + Re-label	**0.850 (0.002)**	**0.924 (0.004)**	**0.929 (0.005)**	**0.870 (0.000)**	**0.678 (0.023)**	**0.330 (0.005)**	**0.558 (0.005)**	**0.593 (0.005)**
Best of Design Space	**0.850 (0.002)**	**0.926 (0.004)**	**0.929 (0.005)**	**0.880 (0.000)**	**0.677 (0.016)**	**0.394 (0.006)**	**0.561 (0.005)**	**0.605 (0.006)**
COSINE (MLP) + Snorkel	0.833 (0.001)	0.915 (0.002)	**0.921 (0.002)**	0.853 (0.002)	0.641 (0.012)	0.248 (0.028)	0.536 (0.004)	0.520 (0.027)
Denoise (MLP) + Snorkel	0.806 (0.011)	0.895 (0.006)	0.528 (0.000)	0.856 (0.001)	0.568 (0.003)	0.169 (0.207)	0.535 (0.004)	0.569 (0.007)
Weasel (MLP)	0.826 (0.003)	0.905 (0.005)	**0.921 (0.016)**	0.856 (0.005)	0.446 (0.117)	0.261 (0.017)	**0.551 (0.007)**	0.552 (0.051)

Table 6. Test performance for LM = Snorkel, DM = MLP. Blue referes to the best performance of a single method (i.e., excluding "Best of Design Space", which reports the best method *per dataset*) while bold refers to being within standard deviation error bars of the best method. Red indicates where "Best of Design Space" outperforms the best single method outside of error bars.

Method	IMDb	Yelp	Youtube	AGNews	TREC	Spouse	Chemprot
Vanilla	0.873 (0.001)	0.919 (0.009)	0.944 (0.007)	0.870 (0.003)	0.656 (0.017)	0.273 (0.074)	0.569 (0.001)
Cutstat + Snorkel	**0.869 (0.015)**	**0.943 (0.006)**	**0.953 (0.002)**	**0.882 (0.001)**	**0.703 (0.017)**	0.242 (0.016)	**0.584 (0.004)**
Conf + ST (DM as LF) + Re-label	**0.888 (0.013)**	**0.954 (0.006)**	**0.947 (0.012)**	**0.882 (0.002)**	**0.711 (0.030)**	**0.531 (0.039)**	**0.581 (0.012)**
Best of Design Space	**0.907 (0.004)**	**0.961 (0.002)**	**0.952 (0.006)**	**0.885 (0.002)**	**0.765 (0.012)**	**0.531 (0.039)**	**0.601 (0.008)**
COSINE (RoBERTa) + Snorkel	**0.875 (0.008)**	0.942 (0.001)	**0.944 (0.010)**	0.875 (0.002)	0.675 (0.002)	0.311 (0.048)	**0.580 (0.005)**
Denoise (RoBERTa) + Snorkel	0.867 (0.007)	0.922 (0.018)	**0.949 (0.006)**	0.867 (0.008)	0.633 (0.014)	0.287 (0.202)	0.544 (0.005)
Weasel (RoBERTa)	0.866 (0.006)	**0.926 (0.028)**	**0.960 (0.008)**	0.873 (0.005)	0.647 (0.021)	0.231 (0.028)	**0.584 (0.014)**

Table 7. Test performance for LM = Snorkel, DM = RoBERTa. Blue referes to the best performance of a single method (i.e., excluding "Best of Design Space", which reports the best method *per dataset*) while bold refers to being within standard deviation error bars of the best method. Red indicates where "Best of Design Space" outperforms the best single method outside of error bars.

Method	IMDb	Yelp	Youtube	AGNews	TREC	Spouse	Chemprot
Vanilla	0.862 (0.010)	0.906 (0.020)	**0.959 (0.010)**	0.870 (0.004)	0.667 (0.013)	0.247 (0.049)	0.572 (0.010)
Cutstat + MV	0.859 (0.006)	**0.954 (0.000)**	0.957 (0.007)	**0.871 (0.008)**	**0.721 (0.023)**	0.244 (0.018)	0.589 (0.002)
Cutstat + ST (DM as LF) + Re-label	**0.882 (0.007)**	**0.954 (0.004)**	**0.969 (0.004)**	**0.882 (0.004)**	**0.743 (0.049)**	0.347 (0.052)	**0.606 (0.004)**
COSINE (RoBERTa) + MV	**0.882 (0.007)**	0.945 (0.004)	**0.956 (0.009)**	**0.876 (0.002)**	**0.702 (0.018)**	**0.431 (0.034)**	0.589 (0.011)
Denoise + MV	0.866 (0.013)	0.905 (0.029)	**0.956 (0.010)**	**0.873 (0.006)**	**0.678 (0.025)**	0.172 (0.205)	0.576 (0.010)
Weasel (RoBERTa)	0.866 (0.006)	0.926 (0.028)	**0.960 (0.008)**	**0.873 (0.005)**	0.647 (0.021)	0.231 (0.028)	0.584 (0.014)

Table 8. Test performance for LM = Majority Voting, DM = RoBERTa.

C. Additional Ablation Results

C.1. Axis Ablation for MLP

Method	IMDb	Yelp	Youtube	AGNews	TREC	Spouse	Chemprot	Census	Mean
Vanilla	0.828 (0.001)	0.917 (0.001)	0.925 (0.004)	0.854 (0.002)	0.623 (0.004)	0.228 (0.013)	0.544 (0.002)	0.511 (0.012)	0.679 (0.002)
Thresh Alone	0.833 (0.002)	0.924 (0.002)	0.915 (0.002)	0.865 (0.001)	**0.678 (0.027)**	0.228 (0.003)	0.547 (0.004)	0.587 (0.001)	0.697 (0.003)
SSL Alone	0.841 (0.004)	0.918 (0.002)	**0.932 (0.000)**	0.877 (0.001)	0.605 (0.008)	**0.375 (0.029)**	0.533 (0.003)	0.518 (0.015)	0.700 (0.004)
Re-label Alone	0.833 (0.001)	0.924 (0.002)	**0.917 (0.014)**	0.862 (0.001)	0.631 (0.024)	0.243 (0.003)	0.549 (0.007)	0.526 (0.027)	0.686 (0.005)
Thresh + SSL	0.838 (0.004)	**0.926 (0.004)**	**0.929 (0.002)**	0.878 (0.001)	**0.691 (0.012)**	**0.394 (0.006)**	**0.560 (0.001)**	**0.605 (0.006)**	0.728 (0.002)
Thresh + Re-label	0.837 (0.002)	0.925 (0.003)	0.917 (0.007)	0.870 (0.001)	**0.677 (0.016)**	0.270 (0.007)	**0.558 (0.002)**	0.602 (0.002)	0.707 (0.002)
SSL + Re-label	0.835 (0.001)	**0.925 (0.006)**	**0.935 (0.005)**	0.875 (0.002)	0.662 (0.011)	**0.347 (0.043)**	**0.557 (0.004)**	0.542 (0.025)	0.710 (0.006)
Thresh + SSL + Re-label	**0.850 (0.002)**	**0.930 (0.001)**	**0.929 (0.005)**	**0.880 (0.000)**	**0.671 (0.034)**	**0.347 (0.043)**	**0.561 (0.005)**	**0.606 (0.001)**	0.722 (0.007)
Entire Design Space	**0.850 (0.002)**	**0.926 (0.004)**	**0.929 (0.005)**	0.880 (0.000)	**0.677 (0.016)**	**0.394 (0.006)**	**0.561 (0.005)**	**0.605 (0.006)**	0.728 (0.002)

Table 9. Test performance for LM = Snorkel, DM = MLP. Each row corresponds to picking a specific method within our design space using validation tuning. Numbers in blue are the highest for any given dataset, while numbers in bold are those within error bars of the best result.

C.2. Axis Instantiations for RoBERTa

	IMDb	Yelp	AGNews	TREC	Spouse	Chemprot
Best w/o SSL	**0.906 (0.005)**	**0.961 (0.002)**	**0.884 (0.003)**	**0.748 (0.023)**	0.336 (0.095)	**0.600 (0.006)**
Best w/ EM	**0.893 (0.011)**	0.956 (0.001)	**0.885 (0.001)**	**0.746 (0.043)**	0.283 (0.083)	**0.601 (0.009)**
Best w/ VAT	**0.900 (0.005)**	0.952 (0.005)	**0.885 (0.003)**	**0.753 (0.021)**	0.244 (0.040)	**0.608 (0.003)**
Best w/ ST	**0.907 (0.004)**	**0.961 (0.003)**	**0.887 (0.005)**	**0.765 (0.012)**	0.392 (0.098)	**0.601 (0.005)**
Best w/ ST (DM as LF)	0.889 (0.004)	**0.958 (0.003)**	**0.885 (0.003)**	0.725 (0.018)	**0.531 (0.039)**	**0.601 (0.008)**

Table 10. Details of the SSL axis for DM = RoBERTa. We report the highest performing method that incorporates a given SSL technique, selected by validation performance.

	IMDb	Yelp	AGNews	TREC	Spouse	Chemprot
No Re-labeling	0.885 (0.018)	**0.959 (0.002)**	**0.887 (0.005)**	0.732 (0.014)	**0.529 (0.057)**	**0.592 (0.005)**
Re-labeling	**0.907 (0.004)**	**0.961 (0.002)**	**0.885 (0.003)**	**0.765 (0.012)**	**0.532 (0.039)**	**0.601 (0.005)**

Table 11. Details on whether re-labeling is useful for DM = RoBERTa. We report the highest performing method that either incorporates or does not incorporate re-labeling, selected by validation performance.

C.2.1. Axis Instantiations for MLPs

Method	IMDb	Yelp	Youtube	AGNews	TREC	Spouse	Chemprot	Census
Cov	0.841 (0.004)	**0.925 (0.006)**	**0.935 (0.005)**	0.877 (0.001)	0.662 (0.011)	0.375 (0.029)	0.557 (0.004)	0.542 (0.025)
Cut	0.832 (0.002)	**0.930 (0.001)**	**0.935 (0.002)**	**0.880 (0.000)**	**0.684 (0.016)**	**0.394 (0.015)**	**0.566 (0.003)**	**0.606 (0.001)**
Conf	**0.850 (0.002)**	**0.928 (0.002)**	**0.929 (0.005)**	0.878 (0.002)	**0.691 (0.012)**	**0.417 (0.009)**	**0.560 (0.011)**	0.592 (0.008)

Table 12. Details of the thresholding axis for DM = MLP. "Best with" represents the highest performing method that incorporates a given thresholding method, selected by validation tuning.

Method	IMDb	Yelp	Youtube	AGNews	TREC	Spouse	Chemprot	Census
Best w/o SSL	0.837 (0.002)	0.927 (0.001)	**0.935 (0.002)**	0.870 (0.001)	**0.684 (0.016)**	0.270 (0.007)	**0.560 (0.011)**	0.602 (0.002)
Best w/ VAT	**0.850 (0.002)**	0.926 (0.002)	**0.935 (0.005)**	**0.880 (0.000)**	**0.691 (0.012)**	0.355 (0.015)	**0.563 (0.007)**	**0.605 (0.006)**
Best w/ EM	0.835 (0.001)	0.922 (0.001)	0.925 (0.002)	0.869 (0.002)	0.664 (0.007)	0.268 (0.032)	**0.560 (0.009)**	0.595 (0.004)
Best w/ ST	0.838 (0.001)	**0.928 (0.002)**	**0.929 (0.005)**	0.866 (0.001)	**0.674 (0.018)**	0.379 (0.008)	**0.566 (0.003)**	0.592 (0.002)
Best w/ ST (DM as LF)	0.834 (0.001)	**0.930 (0.001)**	**0.935 (0.002)**	0.869 (0.000)	**0.668 (0.038)**	**0.417 (0.009)**	**0.561 (0.005)**	**0.606 (0.001)**

Table 13. Details of the SSL axis for DM = MLP. We report the highest performing method that incorporates a given SSL method, selected by validation tuning.

D. Beyond standard WS benchmarks: datasets for coverage ablations

D.1. LF subsampling and procedural generation on existing WRENCH datasets

Overall, WS benchmarks lack agreed-upon ways to explore different LF coverage/precision trade-offs. In this work, we choose to do so by sub-sampling LFs from an overall base set. However, the default LF sets associated with WS benchmarks are not equally as suitable to subsample from. In particular, we are wary about subsampling default LF sets when they:

(1) Contain very few LFs, which reduces the possible granularities of subsampling

(2) Contain LFs which are heterogeneous or have untracked lineages (e.g., λ_1 closely relates to or was only provided in response to another LF λ_2).

For datsets where we subsample directly, we choose ones with both larger LF counts and where all LFs are of the same form. This includes:

- *Semeval:* 164 string-matching rules selected by humans from an automated candidate generation procedure (Zhou et al., 2020)
- *Chemprot:* 26 individual keyword-based rules (Yu et al., 2021)

These contrast with LF sets that are far smaller, more heterogeneous, or have unclear dependencies. For these datasets, we instead procedurally generate LFs using the tools from WRENCH (Zhang et al., 2021)

- *IMDb:* 4 aggregate keyword based rules (i.e., each LF checks for the presence of any of multiple keywords), 1 expression-based rule (Ren et al., 2020)
- *Yelp:* 7 heuristic rules on keywords, 1 third-party model on polarity of sentiment (Ren et al., 2020)
- *Agnews:* 9 aggregate keyword based rules split amongst four respective classes (Ren et al., 2020)

Subsampling. On some datasets, we directly subsample the LFs coming from WRENCH, selecting the most accurate LFs in a class-stratified fashion in order to roughly preserve similar ratios of LFs between classes. We also take care not to completely remove all the LFs for a given class, setting the minimum count to 1 per class. We explore subsampling per-class ratios within $[0.1, 0.9]$, avoiding LF sets that are near-duplicates of each other (i.e., having coverage levels within 1% of each other). We subsample based on accuracy to (optimistically) simulate an LF writer who is both careful and has considerable domain expertise; we assume that if they were to end up at a lower coverage LF set (either by writing fewer LFs or by pruning LFs written during a "brainstorming" phase), they would prioritize rules that they are most confident about. Assuming the LF writer has sufficient domain knowledge, these LFs are also the ones more likely to be accurate on the examples they fire on.

Procedural Generation. For the datasets where subsampling is less appropriate, we use WRENCH's LF generator to construct LF sets with $\{2, 5, 10, 15, 20\}$ n-gram based LFs per class, choosing the most accurate LFs from the candidate pool that have at least 2% coverage over the training set.

D.2. New WS benchmarks: Massive18 and Banking77

We also create two new WS benchmarks by writing LFs for the publicly available intent classification datasets *MASSIVE* (FitzGerald et al., 2022) and *Banking77* (Casanueva et al., 2020). These tasks were chosen a high-level to capture some practical challenges that are not as well-represented by current WRENCH tasks; most notably, they contain significantly higher class counts, which makes them more challenging to write LFs for. Roughly speaking, assuming one writes uni-polar LFs (i.e., each LF either votes for a single class or abstains), the form of the vast majority of LFs in WRENCH, the number of LFs needed to reach a certain coverage level will likely need to scale with the number of classes.[3]. Further, because tasks with larger label spaces require writing more LFs, we also view these tasks as being especially relevant for studying *lower coverage* LF sets. We provide more details about the new tasks as follows:

[3]This, of course, also assumes that the coverage of LFs within each class does not change dramatically

Massive18 is derived from the MASSIVE dataset, a collection of single-shot interactions between human users and general intelligent voice assistants across 52 languages. We exclusively use the English-US portion of this dataset and treat the broad *scenarios* (i.e., general domains) as labels instead of the finer-grained intents. For instance, "`alarm`" is a scenario whereas "`alarm_set`" and "`alarm_remove`" are two intents within that scenario. We call the resulting task *Massive18* since there are 18 different scenario classes. Notably, even with this simplification, *Massive18* contains at least nearly double the class count of all previous WRENCH classification tasks (see Table 14 below).

Banking77 is a collection of online banking queries along with the corresponding user intents. We use this task as defined with the full set of 77 intents, which can be both quite domain specific and especially fine-grained. If we were to write the *minimum* number of LFs for *Banking77*, i.e. one for each class, this would already require 77 LFs, more than the number of LFs for most previous WRENCH datasets.

Datasets	$\lvert\mathcal{Y}\rvert$	|Train|	|Test|	# LFs	Coverage	Snork. Precision	Metric
IMDb	2	20000	2500	5	87.6%	74.4%	Acc.
Yelp	2	30400	3800	8	82.8%	75.4%	Acc.
Youtube	2	1586	250	10	87.7%	87.0%	Acc.
AgNews	4	96000	12000	9	69.1%	82.5%	Acc.
Trec	6	4965	500	68	95.1%	60.0%	Acc.
Massive18 (ours)	**18**	11514	2974	59	61.3%	83.8%	Acc.
Banking77 (ours)	**77**	9003	3080	218	54.5%	76.5%	Acc.
Spouse	2	22254	2701	9	25.8%	65.6% (on Val)	F1 (binary)
Chemprot	10	12861	1607	26	85.6%	58.0%	Acc.
Census	2	10083	16281	83	99.1%	58.0%	F1 (binary)

Table 14. Comparing our two new datasets (blue) to previous benchmarks. Coverage is the percentage of training examples on which at least one LF does not abstain. "Snork. precision" refers to the accuracy of the Snorkel LM on the covered set of inputs.

Labeling Functions. For both datasets, we manually write the LFs ourselves based upon a randomly sampled *development set* of 250 cleanly-labeled examples. Each LF checks whether a specific keyword (or set of multiple keywords) is a substring of the given example and then votes for a specific class if the substring(s) are present (and otherwise abstaining). We provide some examples of the LFs we wrote in Tables 15 and 16. We share the full LF sets in our supplied codebase [4].

Finally, in our experiments, we also try the same subsampling procedure from Appendix D.1 to further explore different coverage levels. For *Massive18* we use the ratios $\{0.1, 0.5, 0.7, 0.8, 1\}$ and for *Banking77* we use the ratios $\{0.1, 0.5, 0.7, 1.0\}$.

Label	Keyword LFs
``alarm''	[``alarm'', ``wake+up'']
``takeaway''	[``takeaway'', ``delivery'', ``order'']
``social''	[``tweet'', ``twitter'', ``facebook'', ``complain'']
``music''	[``what+song'', ``save+song'', ``shuffle'']
``calendar''	[``calendar'', ``schedule'', ``remind'']

Table 15. Example LFs for Massive18. Each row shows one of the possible labels and the associated keyword-based LFs. Note that some LFs contain multiple keywords/substrings that are joined by the "+" character. This signifies that the LF checks whether *all* the keywords supplied are in a given example (though they do not necessarily have to appear in the provided order).

Label	Keyword LFs
``age_limit''	[``age limit'', ``child'', ``my son'', ``my daughter'']
``pin_blocked''	[``takeaway'', ``delivery'', ``order'']
``lost_or_stolen_card''	[``tweet'', ``twitter'', ``facebook'', ``complain'']
``verify_my_identity''	[``id+check'', ``what+ id'']
``disposable_card_limits''	[``disposable+limit'', ``disposable+max'']

Table 16. Example LFs for Banking77. Note the usage of "+", as explained in the caption of Table 15.

[4] https://github.com/jeffreywpli/SSL4WS

D.3. Tabular Datasets

Labeling Functions. We use three tabular tasks and the LF generation procedure from Zhang et al. (2022b). At a high-level, this procedure uses the `sklearn` implementation of random forests (i.e., `RandomForestClassifier`) to train multiple decision trees on a small cleanly-labeled development set (i.e., formed by uniformly sampling 5% of the training set in our experiments). Uni-polar LFs are then derived from individual trees. In our experiments, we generate a set of 100 *candidate* LFs for each task and then select the top-$\{5\%, 10\%, 15\%, 20\%\}$ most accurate LFs (per-class) based upon the full training set (i.e., similar in spirit to WRENCH's procedural LF generator). Again, while this selection step isn't possible without access to all the labels, we consider it as a loose (optimistic) approximation of integrating of a domain expert's knowledge and judgment.

Robustness to seeding. One caveat of this approach for genearting LFs is that the behavior may vary depending on the initial random seed given to `RandomForestClassifier`. To try to account for this, we try multiple seeds and find that the overall conclusions do not change across different runs of the LF generator. As seen in Figure 7, while the absolute performance levels may vary across seeds, SSL is consistently more useful at lower coverage levels.

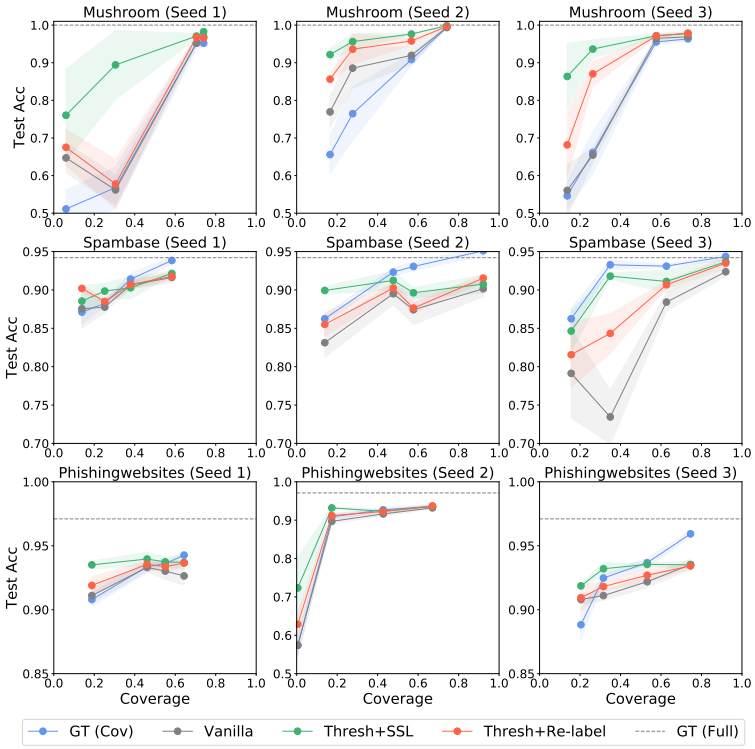


Figure 7. Replication of our tabular experiments across different seeds

E. Coverage bias explains the value of unlabeled data

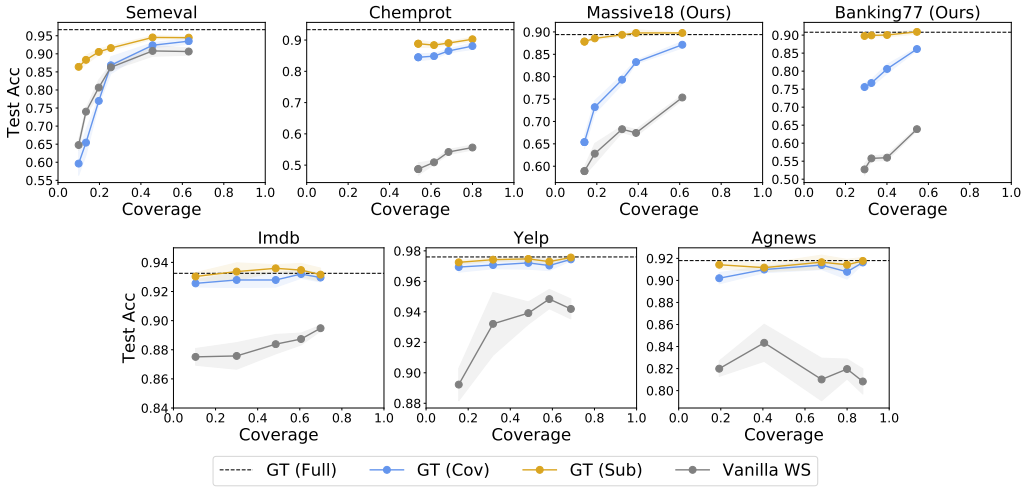


Figure 8. Limited Size versus Coverage Bias. We may assess the relative impacts of the limited size of WS training sets (comparing *GT (Full)* and *GT (Sub)*) versus their coverage bias (comparing *GT (Sub)* and *GT (Cov)*).

In Section 4.5, we observed that on some datasets, as coverage decreases, so does the performance of *GT (Cov)*. Indeed, for any LF set, we can view the gap in performance between *GT (Cov)* and *GT (Full)* as a reflection of the aggregate impact of two *data gaps* that WS datasets suffer from: (1) *limited size* (i.e., not all training examples are labeled) and (2) *coverage bias* (i.e., the examples covered by LFs come from a biased subpopulation of the test distribution). Because *GT (Cov)* simply ignores all unlabeled examples, this gap also then reflects the value of said examples.

A natural follow up question is whether limited size or coverage bias more greatly hinders learning under WS. Here, we further decompose the performance drops we observed between *GT (Cov)* and *GT (Full)* by considering a third model that interpolates between the two. Specifically, we consider *GT (Sub)*, a model that suffers only from limited size but not coverage bias. This model is trained on a training set that shares the same size as the dataset for *GT (Cov)*, but consists of examples *re-sampled uniformly* from the full training set (i.e., eliminating coverage bias).

Once we have trained all three models, we may then simply compare the following performance gaps:

- $\text{TestPerf}(\textit{GT (Full)}) - \text{TestPerf}(\textit{GT (Sub)})$: to ablate the impact limited size
- $\text{TestPerf}(\textit{GT (Sub)}) - \text{TestPerf}(\textit{GT (Cov)})$: to ablate the impact of coverage bias

As we see in Figure 8, when large gaps exist between *GT (Full)* and *GT (Cov)*, most of this gap is explained by the performance drop between *GT (Sub)* and *GT (Cov)* instead of the drop between *GT (Full)* and *GT (Sub)* (e.g., especially on *Semeval, Massive18, Banking77*). This suggests that coverage bias is far more responsible than limited size, which perhaps makes sense in the context of WS, as LFs can label an arbitrary amount of data but only from the covered set. Also, this means that SSL techniques, initially developed in settings where labeled and unlabeled sets are i.i.d. (i.e., where limited size is the only relevant data gap), actually remain effective in low coverage WS settings despite the significant impacts of coverage bias.